#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "s.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//从头部插入节点

void insert\_link\_head(link \*head,link \*new)

{

new=(link\*)malloc(sizeof(link));

new->ch='A';

new->next=head->next;

head->next=new;

}

//从尾部插入节点

void insert\_link\_tail(link \*head,link \*new)

{

new=(link\*)malloc(sizeof(link));

new->ch='C';

link \*p=head;

while(NULL!=p->next)

p=p->next;//遍历到最后的节点

p->next=new;

new->next=NULL;

}

//中间插入节点

void insert\_link\_med(link \*head,link \*new,int n)

{

int i;

link \*p=head;

new=(link\*)malloc(sizeof(link));

new->ch='B';

for(i=0;i<n;i++)

p=p->next;

new ->next=p->next;

p->next=new;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//创建链表

link\* create\_link(char \* a)

{

link \*head;

link \*p=NULL, \*ele = NULL;

int n=0;

n=strlen(a);//获取数个数

//为了避免段错误，在创建链表尽量不要套用函数

if (NULL == ( p=head = (link\*)malloc(sizeof(link)))) //对头结点申请空间

fprintf(stderr, "malloc error\n"), exit(1);

// p=create\_link\_empty(head);error

for ( int i=0;i<n ; i++)

{

if (NULL == (ele = (link\*)malloc(sizeof(link)))) //后续节点申请空间

fprintf(stderr, "malloc error\n"), exit(1);

ele->ch = a[i];

p->next=ele;

p=ele;

}

ele->next=NULL;

return head;

}

//删除指定位置的节点

void del\_link\_ele(link \*head,int n)

{

link \*p=head;//这里p的最终位置是n前面的位置

int i;

for(i=1;i<n;i++)

{

p=p->next;

if(NULL==p->next)

printf("There is no point\n");

}

//1.将结点从链表中摘下来2.手动释放被删除节点

link \*del=p->next;

p->next=p->next->next;

free(del);

}

//删除整个链表=边遍历边删除（只适用于单向）

void del\_them\_all(link \*head)

{

link \*p=head;

while (NULL!=p->next)

{

link \*del=p;

p=p->next;

free(del);

}

}

//遍历

void print\_link(link \*head)

{

link \*p = head;

if (NULL == head->next) return;

while (NULL!=p->next)

{

p = p->next;

putchar(p->ch);

}

putchar('\n');

}

//链表逆序（翻转）（逆置）！！！！！！面试必考！！！！！！！！

//逆序法1:头插法(又名非迭代法)：不断的将1后面的节点插入到head后面

void reverse\_link(link \*head)

{

link \*pele=NULL,\*p=head->next;//p这时指向的是第一个有效节点

if(NULL==p||NULL==p->next) return;

//1.打破锁链

head->next=NULL;//这时头结点指NULL,原有的链表开始重新构造

while (NULL!=p)

{

//2.提取节点

pele=p;//保存p这时的值，第一次以及后面的部分依然是连在一起的

p=p->next;//前进

//3.头节点插入

pele->next=head->next;

head->next=pele;

}

}

//逆序法2 就地逆置法(迭代法)

link\* reverse\_link\_recursion(link \*head) //指向头结点

{

//边界条件:链表为空直接返回，而p->next为空是递归基

if(NULL==head||NULL==head->next) return head;

link \*new=reverse\_link\_recursion(head->next);

if(0!=(head->ch))//p是否到了原来头节点的位置

{

head->next->next=head;//翻转链表的指向!

head->next=NULL;//记得赋值NULL，防止链表错乱

return new;//新链表头永远指向的是原链表的链尾

}

else

{

head->next=new;//建立新的头结点

return head;

}

}

/\*

int main()

{

link \*pH = NULL,\*insert\_ele=NULL,\*newpH=NULL;

char a[5]="12345";

pH=create\_link(a);

printf("original link is: ");

print\_link(pH);

//头部插入，打印输出

printf("insert after head, the link is: ");

insert\_link\_head(pH,insert\_ele);

print\_link(pH);

//尾部插入，打印输出

printf("insert behind tail, the link is: ");

insert\_link\_tail(pH,insert\_ele);

print\_link(pH);

//中间第三个插入，打印输出

insert\_link\_med(pH,insert\_ele,3);

printf("insert before 3, the link is: ");

print\_link(pH);

//删除中间第四个

del\_link\_ele(pH,4);

printf("delete 4, the link is: ");

print\_link(pH);

//逆序

newpH=reverse\_link\_recursion(pH);

printf("after reversed, the new link is: ");

print\_link(newpH);

//删除整个链表

del\_them\_all(pH);

return 0;

}

\*/